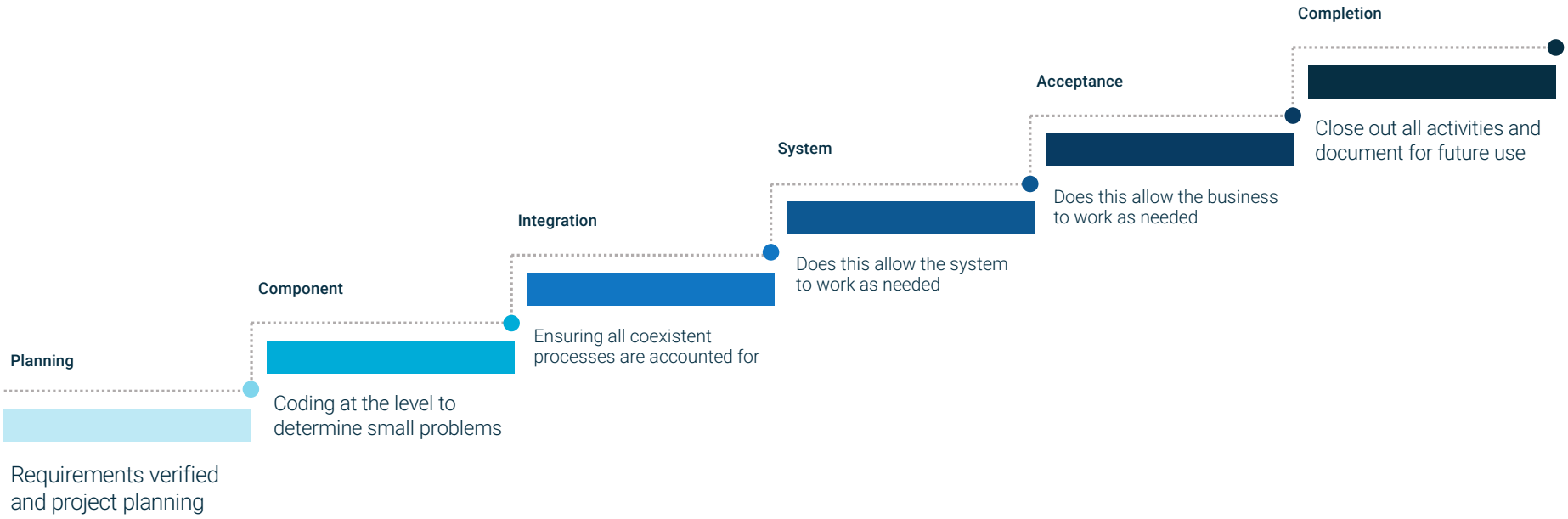
How do I

Improve Software Quality

Scott Klement and Yvonne Enselman

Path for development and testing in unison

Test levels and Sequences



Functional Testing

A request to support a business need was made; does this artifact deliver that functionality as needed.

How the modification proves it does as intended

Component - What was broken out in development for staged improvement and do the calculations or function work?

Component Integration - captured data flow correctly

System - are the approved transactions processing and others being prevented.

System Integration - can the system interact with 3rd party components.

Acceptance - can the user take the provided results and communicate as needed to business and customers.

Non Functional Testing

Can the system perform this task in a safe, reliable, and responsible way

Was the modification developed in keeping with standards for system use

Performance parameters within tolerance

Security vulnerabilities accounted for

Portability and forward maintainability

User experience accepted

Scenario: Code Keeps Breaking

Every time we make a change, it breaks something in production!

```
C     INVHDRK1      SETLL     INVDET
C     INVHDRK1      READE(N)  INVDET                                  10
C                   DOW       *IN10 = *OFF

C                   EVAL      SUBTOTAL = SUBTOTAL
C                                      + %dech(QTY * PRICE: 7: 2)
C                   EVAL      TOTALQTY = TOTALQTY + QTY

C                   EVAL      RRN4 = RRN4 + 1
C                   EVAL      *IN61 = *ON
C                   WRITE     INVENT4S

C     INVHDRK1      READE(N)  INVDET                                  10
C                   ENDDO

C                   EVAL      HIGHRRN4 = RRN4

C                   ENDIF

 * Recalculate total
C                   EVAL      TOTAL = SUBTOTAL + SHIPPING + TAX
```

Smaller Testable Components

This can be called from all programs that need invoice details.

No need to repeat the logic!

It can likewise be called from a Unit Testing Framework.

Each time you make a change, you can test all pieces for regression.

```
count = invoice_getDetail( INV.INVNO
                         : %date(INV.CRTDATE:*iso)
                         : det
                         : %elem(det));
if count = FAIL;
  ERR = INVOICE_getLastErr();
endif;
```

Component Testing

The activity that focuses on why and what is requested at the smallest rational unit of development and testing

Relationship to the requirements is imperative and needs verification

Test Basis - what are the design, code data, and components in play

Test Objects - code and modules being built or modified, organization of data, classifications of both

Typical defects and failures - functions, logic, and data

Ensuring Good Integration

Code should be a “black box”

If the caller knows how it works and does part of the needed logic – it will be hard to reuse, and therefore hard to integrate everywhere it’s needed.

The caller should only need to understand the “interface” (PR or PI)

Use “const”, “varsize”, “omit/nopass” to make it more versatile.

```
dcl-proc invoice_getDetail export;

  dcl-pi *n int(10);
    invno   like(INVHDR_t.invno) const;
    crtdate date(*iso) const options(*omit);
    detail  likeds(INVOICE_detail_t)
            dim(999) options(*omit:*nopass:*varsize);
    detelem int(10) const options(*omit:*nopass);
  end-pi;
```

Integration Testing

The activity that focuses on how what was changed interacts with the existing code and structures in an environment

At this level there can be testing of new processing as well as regression testing based on the requirements

Test Basis - software and system design elements, communications, use cases

Test Objects - Subsystems, databases, APIs, interfaces

Typical defects and failures - Poor data quality in either the test bed or productions, timing issues, communication failures in protocol compatibility.

White Box Testing aka Glass Box

Working with the system directly does the development met criteria for quality.

This is the point to look into the technical processing of the system and break out testing into very technical verification

Complex branch and decision coverage

Data interactions between screens, browsers, and logic

Sequence of processes and opened paths

APIs, micro-services, and other modular interactions

Data formatting for intracompany or business partner exchange interaction.

Change Related

Introducing change into an environment requires standard testing and regression testing in concert

Unknown and previously undiscovered bugs can be a factor in issues found at this point

Automated testing and continuous integration framework is useful

Code repository and change management procedures

Documentation of workflow streams requiring re-testing if anything in the process changes.

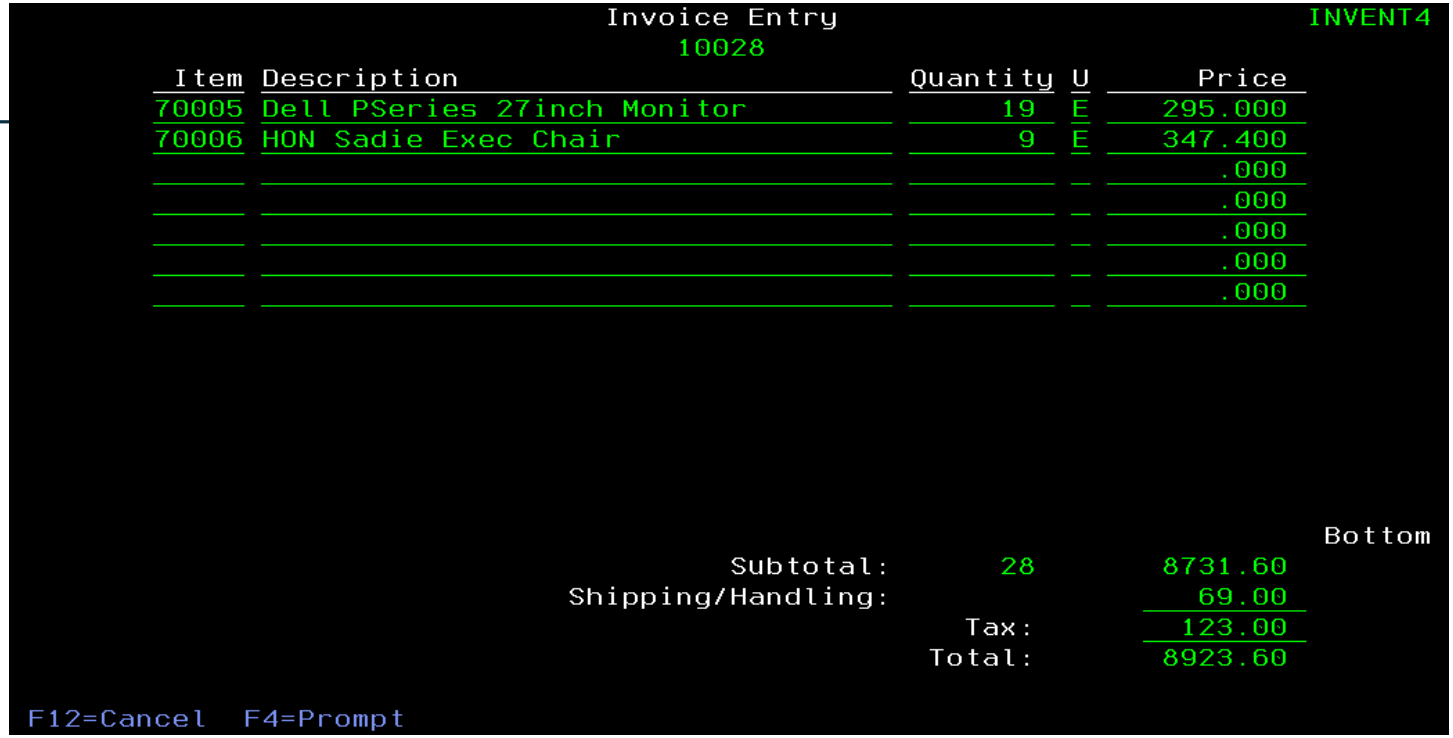
Rerun all failed tests prior to go live to ensure nothing fails if out of sequence

Testing at the System Level

This is where we run the code, make sure it works from the user's perspective.

How do you know if It's designed well?

What makes you confident that you've done it correctly?



performance management APIs

https://www.ibm.com/docs/en/i/7.4?topic=performance-management-apis

System Testing

Focus on ensuring the system can perform the new task as needed and the added processing will not negatively impact the environment as a whole

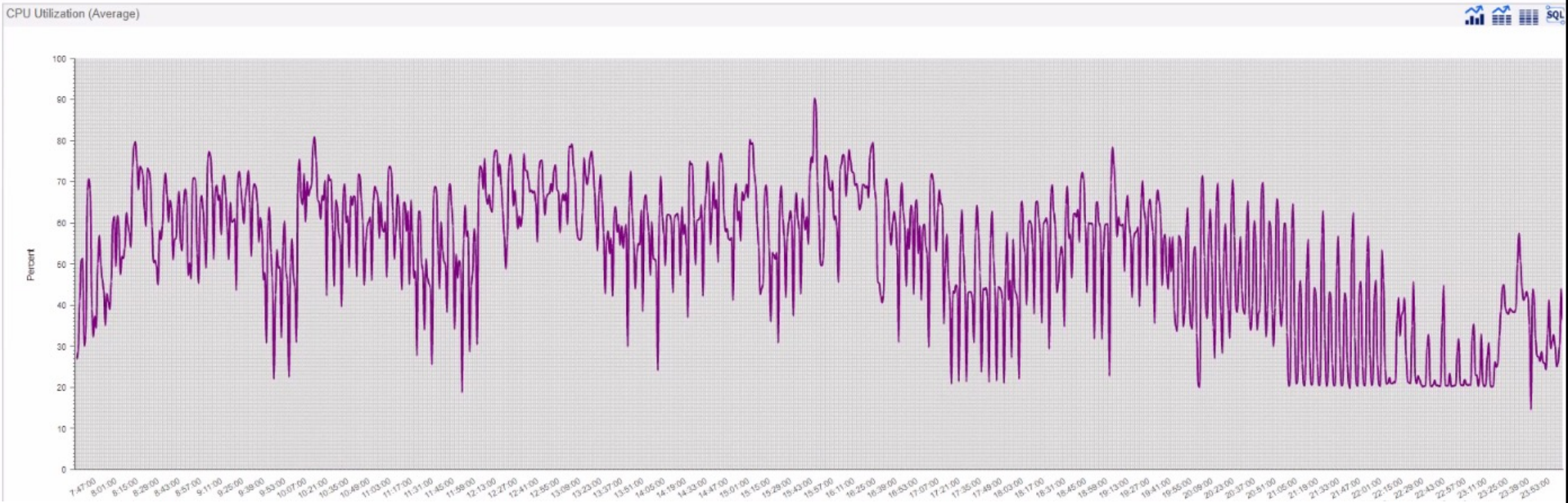
The “ility” testing parts

Test Basis - Requirements and specifications, risk analysis, Epics, models of behavior, state diagrams

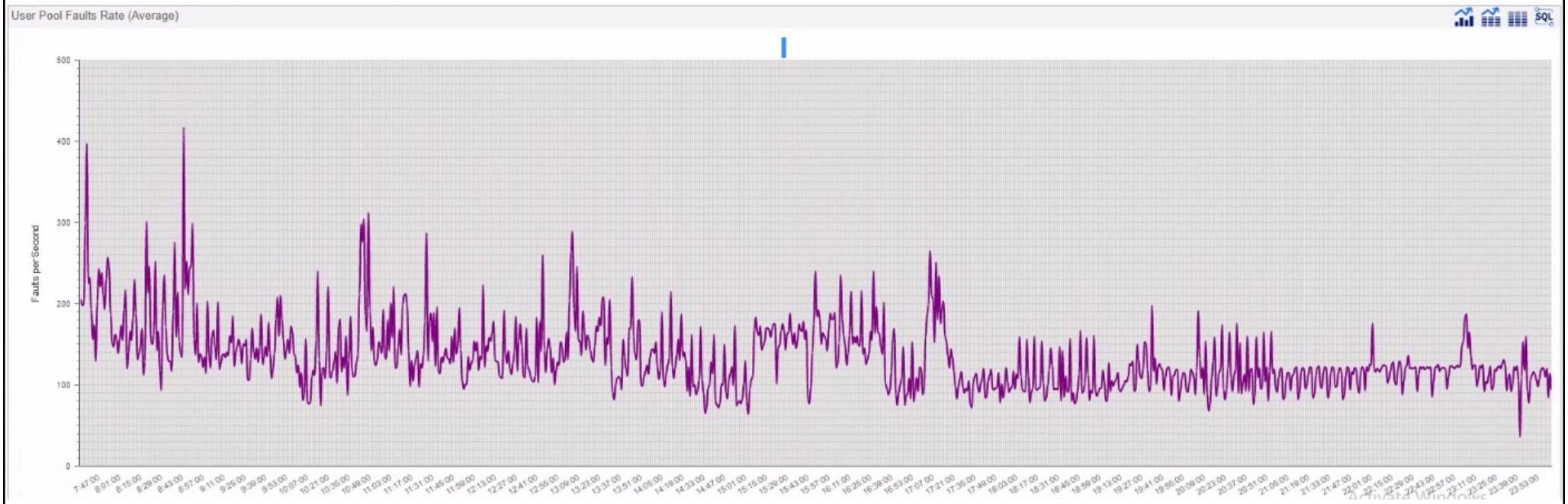
Test Objects - applications, HW/SW systems as a whole. OS, process under test, System configuration

Typical defects and failures - unexpected system performance, security mismatch, data flow concern, User expectations not met

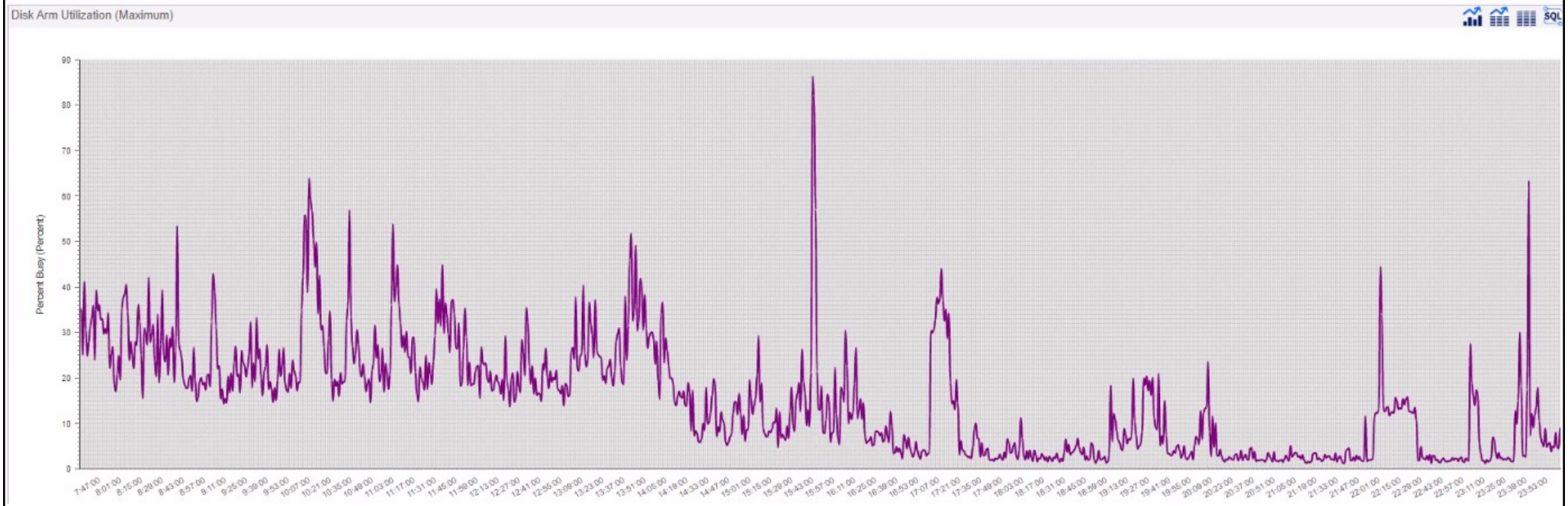
CPU Performance – Efficency



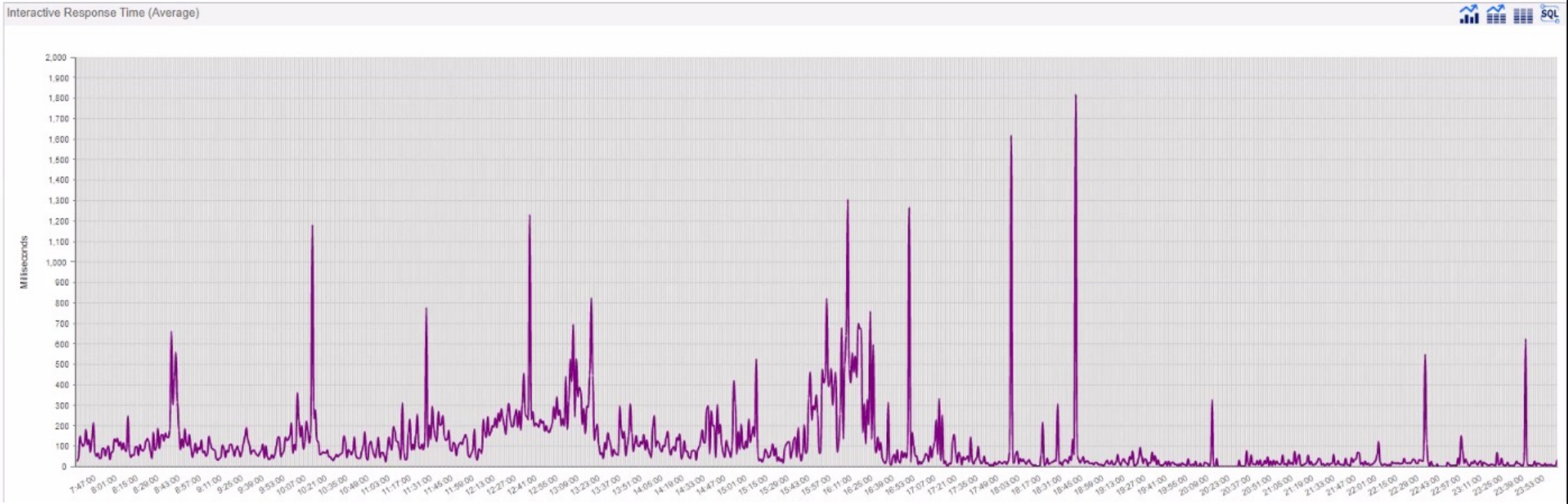
Memory Performance – Reliability



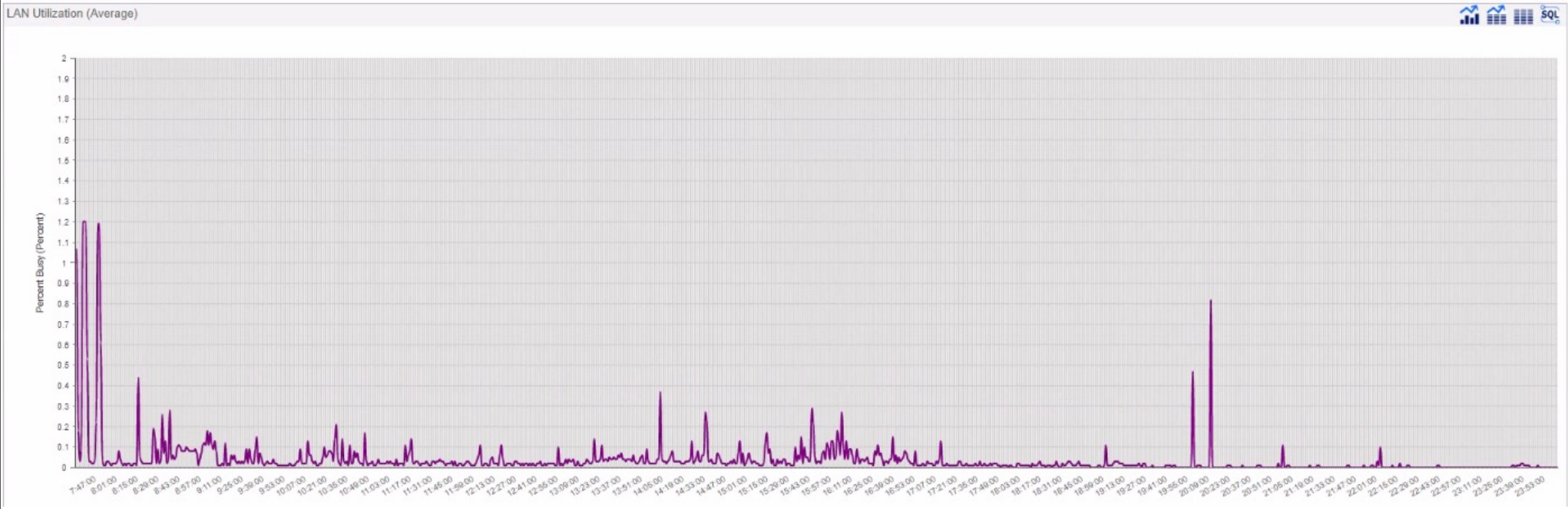
Disk I/O Performance – Useability



Interactive Performance – Portability



LAN Performance – Maintainability



Security

In our system status overview, we do a check of all users with default passwords (user profiles with passwords are equal to the name of the user profile), your security system values, and your file shares.

There are currently **418** ***enabled*** user profiles using default passwords (password = user profile name). This is a ***significant*** security exposure and the recommendation is that all of these profiles have their password changed or should be disabled immediately.

The password level of the system is set to **0**. This limits you to passwords with a max of 10 characters, no mix of upper and lowercase, and limited special characters. We recommend working towards changing this value to at least **2**. Information from IBM on how to plan for this change can be found here: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_73/rzarl/rzarlconsdchg2.htm

There are multiple IFS directory shares (SMB shares) mapped on your system, but no share of **/root** which is good as that greatly mitigates the risk of a ransomware attacks on your system.

Acceptance Testing

Did the modification provide what was requested properly

In System testing we proved the system could function, here we ensure the business can as well.

Test Basis - Business requirements, regulations, user stories

Test Objects - business processing for a fully integrated system, DR/HA site, forms/reports, Production data

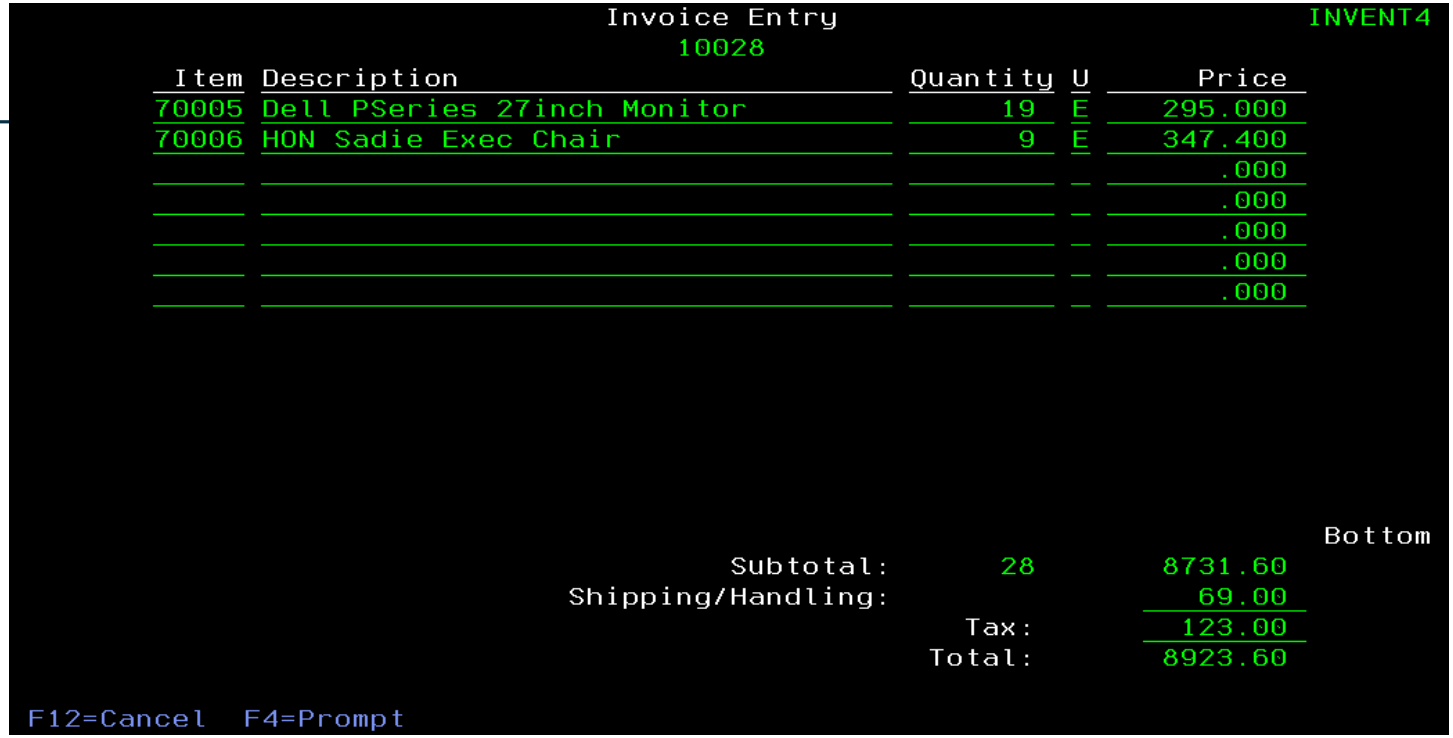
Typical defects and failures - System workflows don’t meet the business or user requirements, regulations were unknown, platform or infrastructure incompatibility.

User Acceptance

Important to have the user try the changes before going live. Let them kick the tires, verify that it works the way they want it to.

The user will see things differently than you do, and will try different things.

The fewer problems found after going live, the better!



Planned Enhancements and Migrations

Keeping environments healthy long term requires updates

Specifications up to date

Document test cases

Bi-Directional traceability needs to be maintained

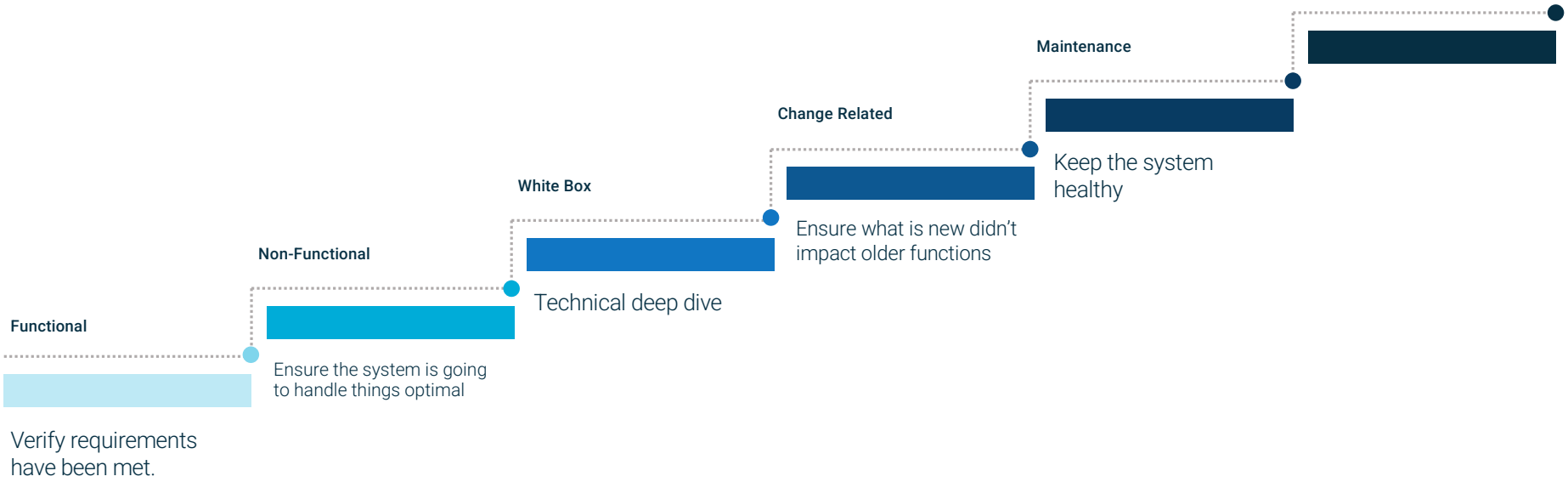
Tool support

People knowledge and contributions

Don’t neglect your code and environments - Technical Debt

Testing Types

All types of tests need to be performed for every level of development



Thank you!